Mining Customer and Item Patterns in Large-Scale Transaction Data Using A-Priori, PCY, and MinHashLSH

*MIDTERM ESSAY

1st Dang Thanh Nhan
Student ID: 522H0006
TON DUC THANG UNIVERSITY
Ho Chi Minh City, Viet Nam

2nd Pham Van Phuc
Student ID: 522H0068
TON DUC THANG UNIVERSITY
Ho Chi Minh City, Viet Nam

3rd Vo Nhat Hao
Student ID: 522H0090
TON DUC THANG UNIVERSITY
Ho Chi Minh City, Viet Nam

4th Nguyen Thanh Nhan
Student ID: 522H0051
TON DUC THANG UNIVERSITY
Ho Chi Minh City, Viet Nam

5th Ngo Duc Huy
Student ID: 522H0038
TON DUC THANG UNIVERSITY
Ho Chi Minh City, Viet Nam

***Abstract*—In this report, our team implemented and analyzed three big data mining algorithms: A-Priori, PCY, and MinHashLSH to solve issues related to mining transaction data. The data used for the project was provided in the baskets.csv file, which contains transaction information from multiple customers. We used Hadoop on an Ubuntu virtual machine and PySpark on Google Colab to implement algorithms.**

The A-Priori algorithm was applied to identify pairs of customers who frequently shop together on the same day, with the implementa-tion carried out through two Map and Reduce steps. The PCY algorithm was used to find pairs of products frequently bought together and to uncover their association rules, using a hash table to reduce the number of pairs of products that need to be checked.

Finally, the MinHashLSH algorithm was implemented to identify pairs of shopping days with products that have a Jaccard similarity above 50%. The results indicate that the algorithms successfully uncovered typical combos of customers and products right after installation.

In addition, the MinHashLSH algorithm helped reduce computation time when comparing the similarity between days, especially under the condition of a Jaccard similarity above 0.5 .This project was completed by our team, fulfilling 100% of the requirements, with results validated through the step-by-step verification of small samples by each team member.

I. Introduction

This project consists of three main tasks to analyze shopping data using data mining algorithms:

- **A-Priori Algorithm for Frequent Customers**
  We will use the A-Priori algorithm to identify frequent customer pairs who shop on the same day. The dataset is processed using Hadoop MapReduce to uncover these customer groups based on a specified support threshold.
- **PCY Algorithm for Frequent Items**
  In this task, we will apply the PCY algorithm to identify frequent pairs of products bought together. The dataset is processed using PySpark DataFrames to efficiently detect item pairs through hashing and bucket management.
- **MinHashLSH for Similar Dates**
  In this task, we apply the MinHashLSH algorithm to identify similar dates based on customer transactions, calculating the Jaccard similarity between items purchased on different days. The task explores two approaches: one using MinHashLSH for faster processing and the other manually comparing all date pairs.

II. DATA INTRODUCTION

A. Dataset Overview

Member_number	Date	itemDescription	year	month	day	day_of_week
1249	1/1/2014	citrus fruit	2014	1	1	2

The data in the file consists of the following columns:

Member_number: This represents the customer ID.

1) **Date**: The date of the transaction in the format dd/mm/yyyy.
2) **itemDescription**: A description of the product purchased.
3) **year**: The year of the transaction.
4) **month**: The month of the transaction.
5) **day**: The day of the month of the transaction.
6) **day_of_week**: The day of the week, with values ranging from 1 to 7.

Each row represents a single product purchased by a customer on a particular day. This data is used to analyze

customer purchases over time, identify products that are frequently purchased together, and identify typical transaction patterns by day or week.

III. A-PRIORI ALGORITHM FOR FREQUENT CUSTOMERS

A. Overview

1) Find Frequent Single Items: Identify all the single items (singletons) whose frequency of occurrence is greater than the support threshold.
2) Generate Larger Itemsets: Generate itemsets from the frequent items in Step 1. Create itemsets of size 2 (pairs of items).
3) Check the Popularity of Itemsets: Check all newly created itemsets in the transactions to identify which ones have a frequency greater than the support threshold. If an itemset is not frequent, discard it immediately because all of its subsets cannot be frequent either.

B. Algorithm Implementation

Pass 1: Identifying High-Frequency Customers
Mapper

Function: Split the data line to extract the customer ID to count the frequent shopping. Emit the pair `<customer, 1>` for each customer.
Input: A line of data containing customer ID and shopping information.
Output: Pair <customer, 1> for each customer.

1) If this is not the header line:
2) output (CustomerID, 1)

Reducer

Function: Calculate the total occurrences of each customer. If the frequency of the customer exceeds the support threshold (`SUPPORT_THRESHOLD`), emit the pair `<customer, frequency>`.
Input: Pairs `<customer, 1>` from the Mapper.
Output: A pair `<customer, frequency>`.

1) sum = 0
2) For each value in List of 1's:
3) sum += 1
4) If sum $\geq$ SUPPORT_THRESHOLD:
5) output (CustomerID, sum)

Pass 2: Implementation
Mapper:

Function: Read data containing customer information and purchase date and output pairs <Purchase Date, Customer ID>
Input:
The data streams contain information about customers and purchase dates and a list of popular customers from the output of
Output:
<Purchase Date, Customer ID> pairs only with popular customers

1) Initialize frequentCustomers as an empty set
2) Read output of Pass 1:
3) For each line in the file:
4) Extract CustomerID
5) Add CustomerID to frequentCustomers
6) For each line in input data:
7) If this is not the header line:
8) Extract CustomerID and Date
9) If CustomerID is in frequentCustomers:
10) Output (Purchase Date, CustomerID)

Reducer:

Function: Calculate the total number of occurrences of each customer pair. If the frequency of this customer pair exceeds the support threshold, output the pair `<customer_pair, frequency>`
Input:
Pairs of dates and customer codes from Mapper
Output:
Pair `<customer_pair, frequency>`

1) Initialize pairCounts as an empty map
2) For each Date received:
3) Collect all CustomerIDs associated with that Date into customerSet
4) For each unique pair of CustomerIDs:
5) If the pair exists in pairCounts:
6) pairCounts[pair] += 1
7) Else:
8) pairCounts[pair] = 1
9) For each pair in pairCounts:
10) If pairCounts[pair] $\geq$ SUPPORT_THRESHOLD:
11) Output (CustomerID1, CustomerID2, pairCounts[pair])

EXAMPLE

Input: taken from the first 7 lines of the provided csv file.
Pass 1:
Mapper:
Each customer emits a pair <customer_id, 1>.
Reducer:

Input	Output
1249, 01/01/2014	(1249, 1)
1249, 01/01/2014	(1249, 1)
1381, 01/01/2014	(1381, 1)
1381, 01/01/2014	(1381, 1)

Calculate the total occurrences of each customer. The support threshold is 2 (meaning each customer needs to appear at least 2 times to be considered frequent).
Customers with a frequency of occurrences >= 2 are:

- 1249 (appears 2 times)
- 1381 (appears 2 times)

Pass 2:
Mapper:
Select only customers in Regular Customers
Create pairs:

Date	CustomerID
01/01/2014	1249
01/01/2014	1249
01/01/2014	1381
01/01/2014	1381
01/01/2014	1440
01/01/2014	1440

Reducer:
Group customers by day.
Ngày 01/01/2024: 1249, 1381, 1440
Create customer pairs by day:
(1249, 1381), (1249, 1440), (1381, 1440)
Update to pairCounts:
pairCounts =
(
(1249, 1381): 1,
(1249, 1440): 1,
(1381, 1440): 1
)
Compare with THRESHOLD level:

- Since each pair appears only once on 01/01/2014, if SUPPORT_THRESHOLD ≥ 2, no pair will be emitted.
- If SUPPORT_THRESHOLD = 1, all pairs will be emitted.

IV. PCY ALGORITHM FOR FREQUENT ITEMS

A. Overview

1) Read input data then group by customer and date to create shopping cart using PySpark DataFrames.
2) Pass 1: Count frequent single items, hash pairs into buckets, create a bitmap for frequent buckets.
3) Pass 2: Count pairs in frequent buckets (bitmap = 1), filter frequent pairs based on support threshold s.
4) Generate association rules from frequent pairs, filter by confidence threshold c.

B. Algorithm Implementation

1. First Pass (Bucket Counting)

Input: All baskets (transactions)
Output: List of buckets and counts, plus bitmap
Function: Identifies buckets containing pairs of items that appear frequently in the transaction data.

1) For each basket:
2) Generate all possible pairs of 2 distinct items in the basket.
3) For each pair:
4) Hash the pair into a bucket (integer value between 0 and numBuckets-1).
5) Increment the count for the corresponding bucket.
6) Filter buckets with counts $\geq$ minSupport $\rightarrow$ frequent buckets.
7) Create a bitmap (binary array) where 1 indicates a frequent bucket, 0 otherwise.

2. Second Pass (Count Candidate Pairs)

Input: All baskets again, bitmap from First Pass
Output: List of pairs and counts
Function: Identifies truly frequent item pairs by using the bitmap from the First Pass to filter out potentially non-relevant pairs.

1) For each basket:
2) Regenerate all possible item pairs.
3) For each pair:
4) Check the bitmap: If the bucket for this pair is marked as frequent:
5) Increment the count for the pair.

3. Generate Association Rules

Input: Frequent pairs, all baskets
Output: List of rules
Function: Generates meaningful association rules (like "If A then B") from the frequent item pairs discovered in previous steps, calculating confidence metrics to determine rule strength. Call CalcItemSupport(baskets) to get support of each item.

1) For each pair (A, B):
2) Calculate support of A (total occurrences of A in all baskets)
3) Calculate support of B (total occurrences of B in all baskets)

4) Confidence of A→B = (support of pair) / (support of A)
5) Confidence of B→A = (support of pair) / (support of B)
6) If confidence of A→B ≥ minConfidence:
7) Add rule: "If A is bought, then B is likely bought"
8) If confidence of B→A ≥ minConfidence:
9) Add rule: "If B is bought, then A is likely bought"

EXAMPLE

Input Data

TABLE I
INPUT DATA

Member_number	Date	itemDescription
1249	1/1/2014	citrus fruit
1249	1/1/2014	coffee
1249	1/2/2014	citrus fruit
1249	1/2/2014	coffee
1381	1/1/2014	curd
1381	1/1/2014	soda

Parameters

- **minSupport = 2**: Pairs must appear at least 2 times.
- **numBuckets = 4**: Number of buckets in hashing.
- **minConfidence = 0.5 (50%)**: Minimum confidence threshold is 50%.

FIRST PASS

TABLE II
PAIRS FORMED FOR EACH CUSTOMER ON 1/1/2014

Member_number	Pairs Formed
1249	(citrus fruit, coffee)
1249	(citrus fruit, coffee)
1381	(curd, soda)

TABLE III
HASHING PAIRS INTO BUCKETS

Pair	Hash Value % 4 (Bucket)
(citrus fruit, coffee)	2
(citrus fruit, coffee)	2
(curd, soda)	1

Bitmap: {2}

SECOND PASS (COUNT CANDIDATE PAIRS)

Test with Bitmap

TABLE VI: BUCKET POPULAR PAIRS AFTER CHECKING WITH BITMAP

TABLE IV
COUNT OF PAIRS IN EACH BUCKET

Bucket	Count
0	0
1	1
2	2
3	0

TABLE V
FREQUENT BUCKETS WITH MINSUPPORT = 2

Bucket	Count
2	2

Pair	HashValue% 4	In Bitmap?	Count
(citrus fruit, coffee)	2	Yes	2
(curd, soda)	1	No	0

Results (Bucket Popular Pairs):
Generate Association Rules (Assume minConfidence = 0.5)
Calculate Support for each item:

TABLE VII: SUPPORT FOR EACH ITEM

Item	Support
citrus fruit	2
coffee	2
curd	1
soda	1

Generate rule

- citrus fruit → coffee

$$\text{Confidence} = \frac{\text{Support(citrus fruit, coffee)}}{\text{Support(citrus fruit)}} = \frac{2}{2} = 1.0 = 100\%$$

- coffee → citrus fruit

$$\text{Confidence} = \frac{\text{Support(citrus fruit, coffee)}}{\text{Support(coffee)}} = \frac{2}{2} = 1.0 = 100\%$$

V. MINHASHLSH FOR SIMILAR DATES

A. Overview

1) **Problem requirements:**
The task uses PySpark DataFrames to find pairs of dates with Jaccard similarity 0.5 based on purchased items. We have applied two approaches: MinHashLSH from pyspark.ml.feature.MinHashLSH and a Brute Force method that checks all date pairs. After execution, the results of the two approaches are compared through running time plots with threshold ranging from 0.0 to 1.0.
2) **MinHashLSH Description:**
The steps of MinHashLSH implementation include:

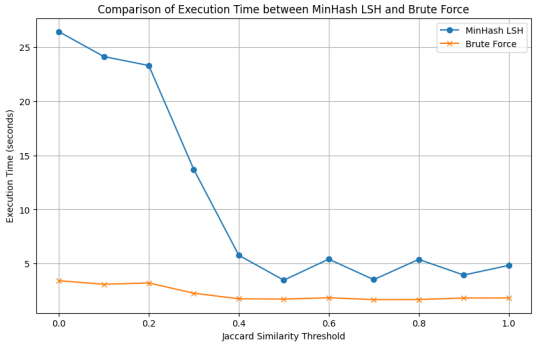
- Convert the item sets of each day (Date) into binary vectors (features) using CountVectorizer.
- Use pyspark.ml.feature.MinHashLSH to generate hash signatures (hashes) that compress the representation of each day.
- Compare the hash signatures to find pairs of dates with high Jaccard similarity.

3) **Brute Force Description:**

- Generate all possible pairs of dates (Date).
- Calculate the Jaccard similarity between each pair of dates by comparing their item sets.
- Keep the pairs with similarity greater than or equal to a specified threshold.

4) **Compare:**

The graph shows the running time of the two approaches applied:



VI. CONCLUSION

- In this study, we implemented and evaluated three data mining algorithms on a large shopping dataset.
- Task 1: We kept the data in HDFS and did the A-Priori algorithm using Hadoop MapReduce (Java) to find out common customer pairs who shopped on the same day. The method had two passes: in the first pass, we noted down frequent customers and in the second pass, generated and filtered customer pairs based on the support threshold. The output was a list of customer pairs who often shopped together.
- Task 2:
  - PCY improves performance by reducing the number of pairs to count in Pass 2 thanks to bitmaps.
  - Efficiency depends on the hash distribution; if the buckets are too small or uneven, bitmaps are ineffective.
  - Performance better than traditional A-Priori, especially when integrated with PySpark.
- Task 3:
  - MinHashLSH and brute-force approaches were applied to discover similar dates based on Jaccard similarity, with performance comparison across thresholds.
  - MinHashLSH is slower than Brute Force at all Jaccard thresholds, but the running time decreases as the threshold increases. Meanwhile, Brute Force does not change much in running time. The reason is that MinHash LSH is efficient with large data, while Brute Force only iterates over pairs without spending time transforming the data.
- The experiments demonstrate the effectiveness of Hadoop MapReduce and PySpark for large-scale data mining, providing insights for improving customer behavior analysis and recommendation systems.

VII. CONTRIBUTIONS

TABLE VIII
SELF-EVALUATION OF MEMBER CONTRIBUTIONS

Name	Contribution (%)
Dang Thanh Nhan	100%
Pham Van Phuc	100%
Vo Nhat Hao	100%
Nguyen Thanh Nhan	100%
Ngo Duc Huy	100%

VIII. SELF-EVALUATION

TABLE IX
SELF-EVALUATION OF TASK COMPLETION AND SCORES

Task	Completion (%)	Max Points
Task 1: A-Priori Algorithm for Frequent Customers	100%	3.0
Task 2: PCY Algorithm for Frequent Items	100%	3.0
Task 3: MinHashLSH for Similar Dates	100%	3.0
Task 4: Report	100%	1.0
Total	–	10.0

REFERENCES

[1] J. Han, M. Kamber, and J. Pei, "Data Mining: Concepts and Techniques," 3rd ed. San Francisco, CA: Morgan Kaufmann, 2011, pp. 243–278. [Online].

[2] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in Proc. 20th Int. Conf. Very Large Data Bases (VLDB), Santiago, Chile, Sep. 1994, pp. 487–499. [Online].

[3] S. Park, J. S. Park, and M.-S. Chen, "Using a hash-based method with transaction trimming for mining association rules," IEEE Trans. Knowl. Data Eng., vol. 9, no. 5, pp. 813–825, Sep./Oct. 1997. [Online].

[4] A. Rajaraman and J. D. Ullman, "Mining of Massive Datasets," Cambridge, UK: Cambridge Univ. Press, 2011, pp. 167–210. [Online].

[5] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in Proc. 30th Annu. ACM Symp. Theory Comput., Dallas, TX, May 1998, pp. 604–613. [Online].

[6] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in Proc. 25th Int. Conf. Very Large Data Bases (VLDB), Edinburgh, Scotland, Sep. 1999, pp. 518–529. [Online].

[7] M. Young, "The Technical Writer's Handbook," Mill Valley, CA: University Science, 1989. [Online].